CS159

Nathan Sprague

September 30, 2013

Recursive Definitions

Merriam Websters definition of Ancestor:

Ancestor

One from whom a person is descended [...]

Here is a recursive version:

Ancestor

One's parent.
or
The parent of one's ancestor.

Recursively Defined Functions

Classic example is the factorial function:

$n!$

if $n = 0$ then $n! = 1$ *(basis or initial conditions)*
if $n > 0$ then $n! = n \times (n-1)!$

Recursive Methods / Recursive Programming

A recursive method is a method that includes a call to itself. It is often straightforward to compute recursively defined functions using recursive methods:

```
1  int factorial(int n)
2  {
3      int value;
4
5      if    (n == 0)
6          value = 1;
7      else
8          value = n * factorial(n - 1);
9
10     return value;
11 }
```

Activation Records

Every method call results in an activation record which contains:

- Local variables and their values.
- The location (in the caller) of the call.

Tracing Recursive Methods...

Recursion is Not Always the Best Approach

```
1  int factorial(int n)
2  {
3      int value = 1;
4
5      for (int i=2; i <= n; i++)
6      {
7          value *= i;
8      }
9
10     return value;
11 }
```

Recursive Problem Solving

Recursion is often a good idea when a problem can be solved by breaking it into one or more smaller problems of the same form. The process is:

- Figure out how to solve the easy case.
- Figure out how to move the hard case toward the easy case.

Recursion Pseudocode

Nearly every recursive method ends up looking like the following:

```
1
2   recursiveMethod(input)
3   {
4         if (input represents an easy case)
5         {
6               handle the easy case directly.
7         }
8         else
9         {
10              call recursiveMethod one or more times
11              passing it only part of the input.
12        }
13  }
```

The Coin Problem

Determine the minimum number of coins needed to make change for a given amount.

- The easy case:
  - We can use a single coin.
- Reducing the hard case:
  - Try every way of splitting the amount into two parts: j and amount - j
  - recursively find minimum coin solution for each pair
  - return the minimum.

(Note... this is really slow.)